FileSafe

Fiona Gaugush, Shawn Shao, Tramy Dong, Sarah Aktari

BCCOMS3997: Building Usable Security

Section 1. Executive Summary

FileSafe is a secure mobile application designed to help users safely store and have easy access to their sensitive documents. This app was designed to tackle the issue of security of personal documents containing sensitive information. It would be dangerous to keep the physical copies of these documents, but saving them through email or notes apps also poses the risk of a data leak. *FileSafe* tackles this issue by encrypting and storing files such as passports, tickets, and IDs as well as allowing users to write information down in our notes feature. We drew inspiration based on a few existing applications such as TripIt and Apple Notes with all offer document storage features. However, FileSafe prioritizes user-security through strong encryption.

The app is designed to be used by a range of target audiences, some major ones being: international students, people applying for jobs, and non-tech-savvy individuals. Users can log in using any of the following four options: Google, Apple, phone number, or for extra security, an anonymous login option which limits the leak of personal information such as emails and phone numbers. Uploaded documents are encrypted using a user-selected PIN then the encryption process secured through AES-GCM encryption and stored in Firebase.The process is as follows: the encryption function takes the user selected passcode which is converted to the form UTF-8 using the function .data(using: .utf8). A symmetric key for the password is then combined with a generated salt and hash using SHA256, resulting in the encryption key in the system. The function generates a nounce using AES.GCM.Nounce and uses AES.GCM to encrypt the key and nounce. The result is set to be a sealedBox containing the encrypted data. This employs Firebase Authentication to manage user credentials and encrypted data. Additional security features include Face ID and multi-login recovery options. However, risks include user dependence on digital access, potential account lockout, and exposure to threats like shoulder surfing or third-party breaches. Usable features on this app include photo/file uploads, and new notes uploads which get sent to a directory and can be retrieved using a PIN. Our functionality non-goals included AI integration and auto-deletion of files at a certain expiration date, but we focused on the goals mentioned previously. There are user trade-offs we had considered as well: mainly that people are dependent on using the notes app features already provided on their phones. Additionally the requirement of both password and PIN entry leads to higher risk for forgotten credentials which could result in lost document access. As such, we conducted usability testing through peer feedback and cognitive walkthroughs to better understand the parts of the application that would be benefited with improvement and which features worked smoothly. The participants enjoyed the UI aesthetic, and suggested improvements to UI features (changing button color to indicate that you can press it now, hiding password with asterisks). The biggest takeaway was that although people understood the link between our app and security, they did not understand how encryption made documents secure. After explaining what encryption, users mentioned they were more inclined to use it to store important information as opposed to the default Notes app on their phone. Thus we were inclined to add an FAQ section to make this app more accessible and would push users to use it.

Section 2. Project Overview

The primary goal of *FileSafe* is to store and protect users' documents. We aim for this to be both secure and easy to navigate. This serves as an alternative to storing files or information that needs to be accessed regularly in the notes app or in the camera roll. If multiple people have access to this account, the different tiers of PINs allow different tiers of access for each person. Users are given the option to log in with Gmail, phone number, or Apple which limits the amount of passwords they need to remember allowing ease in use. Users are however asked to create and remember a default PIN which will encrypt their files and without this PIN, any files saved will be lost. Although there are shortcomings in the ability for users to recover files, *FileSafe* offers a streamlined and secure way of file storing.

Section 2.1. Motivation and Use Cases

Use Case 1: International Students traveling between school/home. In this case, international students who have to travel frequently between home and school can use this app to store their documents. This could include vaccination records, passports, tickets, proof of enrollment, etc.

Use Case 2: Individuals traveling for work. For this case, users are traveling frequently, and already have the stress of work to be concerned about. They can have all the documents they use frequently in one place that they can access at any point. They can include their government travel documents as well as important information surrounding the nature of their business.

Use Case 3: Individuals traveling abroad for leisure. People traveling leisurely can keep all their documents related to travel in one place without worrying about their digital security in places they may not know very well. If going to multiple locations, they can sort what documents they need for each location accordingly.

Use Case 4: Individuals who need to store important documents, but are not too tech savvy (ex: elders/children/their family members). In this specific case, it would be important to have all important documents in one place. *FileSafe* provides a way for those people's caretakers/family members to load their important documents onto this app for them, so they do not have to worry about losing physical copies or not finding the digital ones in a disorganized camera roll/notes app.

Use Case 5: Keeping relevant documents for employment. This case would include people who constantly need the same documents on hand, whether that be for applications or onboarding. The information on the documents is more important than the actual document (SSN, ID number, records of previous employment) for these users, so *FileSafe* serves as an organized tool for them.

Use Case 6: Keeping relevant documents for medical reasons. This case would include situations where people need to access information about their medical history. This could be to physically show doctors or to keep for their own knowledge (medications, diagnosis, receipts/bills).

Section 2.2 Related Works

Notes [1]
Notes app was a major inspiration for this project due to its frequent use as a default app on iPhones. On this app users can scan documents, take photos of documents, and type and draw. Notes can be grouped together by folders and notes. These notes can be locked and encrypted. This aspect is similar to our app, however every document will be encrypted and password protected (the app itself will also be password protected).

TripIt [2]
TripIt is directly relevant to our motivating usecase of travel. TripIt focuses on many different aspects of travel including help with planning an itinerary and getting directions. Similar to our proposed app, they have a feature to store PDFS, photos, boarding passes, etc. *FileSafe* will focus primarily on the safety of these documents rather than the other aspects of travel. To get to important documents like passports on both our app and TripIt, a password must be entered. Both apps encrypt these documents.

Documents: File Manager & Docs [3]
Documents is an app used for managing documents, but has such a wide range of features including acting as a player, editor, vpn, and transferring. This app can connect to many different file sharing/storing apps (Photos, DropBox, etc). Our app is similar as it allows users to store their documents in convenient folders. Documents can be password protected, however you can password protect documents outside of the ones loaded on in the app. This app is more geared to storing a large range of documents rather than just focusing on the security.

Section 3. Requirements

Section 3.1. Brief System Summary

FileSafe is a mobile application. Users can use Google/Apple/phone number or set their own username to log into this app. Users can use four ways to encrypt: taking a photo using the camera, uploading photos, uploading any form documents and type notes. All of these things will be encrypted by a PIN set by the user and Firebase will store these encrypted files. Users can also set additional security for each individual file by either using an additional PIN to encrypt the file or using FaceID.

Section 3.2. Functionality Goals

Secure Login Flow. Users will be able to register and use Google/Apple/Phone/Any username+passcode to login to the app. These login credentials will be stored securely supported by Firebase.
Anonymous login option. Users can choose to anonymously login by choosing the username+password combination, in this way, users will not provide any real information about accounts.

Secure storage of encrypted files. All the encrypted files will be stored securely in Firestore internally. If we, as developers, were to try to download the files from Firestore, we could not see the contents because they are encrypted. Externally they are also protected because our Firebase account is private (and even if it wasn't the files could not be viewed).

User-friendly encryption/decryption process. We choose user-friendly UI to make the encryption flow and decrypt flow be very easy to use and understand

Take/choose photos. Users can choose to encrypt files whenever they want by photographing what they want to be encrypted. Along with that, our app also provides the functionality of cropping the document to help users select the parts they want.

Write Notes. Users can always open the app to encrypt their notes

Choose Files. Users can choose the files in their phone to encrypt.

Organizational flexibility. Users can create self-named directories like medical documents and put documents into these folders to help them better organize their encrypted files.

Renaming the encrypted files. When users view their decrypted files, they can rename the files they want.

Additional Security for files: We will provide additional security practices including additional passcode and face ID for users to create additional secure practices to protect their files.

Complete deletion for files. Users can swipe in the decrypt view and choose to delete the file to completely delete the file information in Firebase.

A detailed walkthrough tutorial. A tutorial including any complicated step involved in the encrypting or decrypting process will be displayed at the first time when user logs in, and after that, there will be a button in the main view that users can click to review the tutorial if they need. There will also be a FAQ and Privacy Statement that users can refer to for more information.

Additional security for login. We set up a security button in the setting for users to connect two login methods together (Phone + Email) so that if a user forgets one login method, they can still access their accounts using another method.

Section 3.3. Functionality Non-goals

Sharing Feature. The app does not allow for sharing documents.

Auto Deletion. We will not support auto deletion for files because users may lose the files they want.

Downloading onto the device. Documents are strictly confined within the FileSafe environment with no functionality of directly downloading.

No AI connection. FileSafe will not incorporate any AI connection so AI will not get any personal info about the files.

No leaking any personal information. Our app will not require any personal info like name or profile picture. (The only thing we may collect is your email but you still have the opportunity to choose the anonymous login.)

No recovery for PIN. Our app will not provide a way to reset or recover the PIN if it is lost.

No recovery for anonymous login: Because the purpose of the anonymous login is for the account to not be linked to phone number, email, or Apple, we did not find a way to recover the account if the username or password is lost.

Section 3.4. Threat Model

Section 3.4.1. Actors

1. International Students
    - International students will use our app because they are young and need a place to back up their files especially for first year students.
2. Business travelers
    - Business travelers usually have a lot of files to take during their foreign business trips, so it is very important to have a second place to secure their files in case they forget to bring.
3. Non-tech-savvy users (ex. elders)
    - These users are used to gaining help from their families or friends, so when they travel alone, they may not have the ability to secure their docs, and our app will provide a easy way to help them.
4. People given the information in the files purposefully..

- These are people who the user entrusts with the information that is stored in the documents such as transportation employees, potential employers, etc.

5. Developers
    - Developers will design the whole app, manage all of the users, process all the feedback, and protect all users' encrypted files.
6. Hackers
    - Malicious actors may attempt to breach the app's backend in Firebase to access users' documents in Firebase and use brutal force to decrypt these documents.
7. Thieves
    - These actors target the physical device, attempting to access data when users log in through theft.
8. Network
    - All the files encrypted by users will be connected to Firebase through the network. In the process of transmitting data, the network may be threatened by Hackers.
9. Government
    - Government agencies like police stations may supervise our app to prevent the app from selling users' private information.

Section 3.4.2. Assets

1. Login accounts
    - Our app owns these passcodes to give them to every user to log into their unique accounts to see their encrypted files.
2. FIles/documents
    - Our app owns these Files and documents stored by every user and they are securely stored in Firebase.
3. Decryption PIN
    - Our app owns the decryption PINs to give to every user in the app to decrypt their files.
4. Cloud Storage(Firebase Storage):
    - Our app owns the firebase storage for our account to provide the safe space for users to put their encrypted files.
5. Firebase Authentication tokens:
    - Our app owns the SMS, google, apple login tokens to authenticate users' identity to ensure other people cannot go into their accounts.
6. App UI:
    - Our app owns the UI to be different from other apps and provide users with better experience with our app.
7. Additional Passcode or Face ID:
    - Our app owns these additional secure practices users set to protect their files to give to every user another layer of security for their files.

Section 3.4.3. Security Goals

1. **Exclusive Document Access:** The files encrypted by users can only be accessed by the user who holds the passcode and account. The files cannot be downloaded from Firestore because Firestore only keeps the encrypted versions.
2. **Login Security:** Every account can only be accessed through the user who knows the account's login credentials including gmail, apple, or phone SMS.
3. **Strong and secure encryption for documents:** The encryption process is very complex and we aim that brutal force will take a very long time like several months to get the passcode set by users.
4. **Firebase Data Storage security:** Firebase Security Rules are configured to ensure that only authenticated users can access their own data.
5. **Personal Information security:** we will ensure that no personal information will be leaked such as email or phone numbers.

Section 3.4.4. Security Non-goals

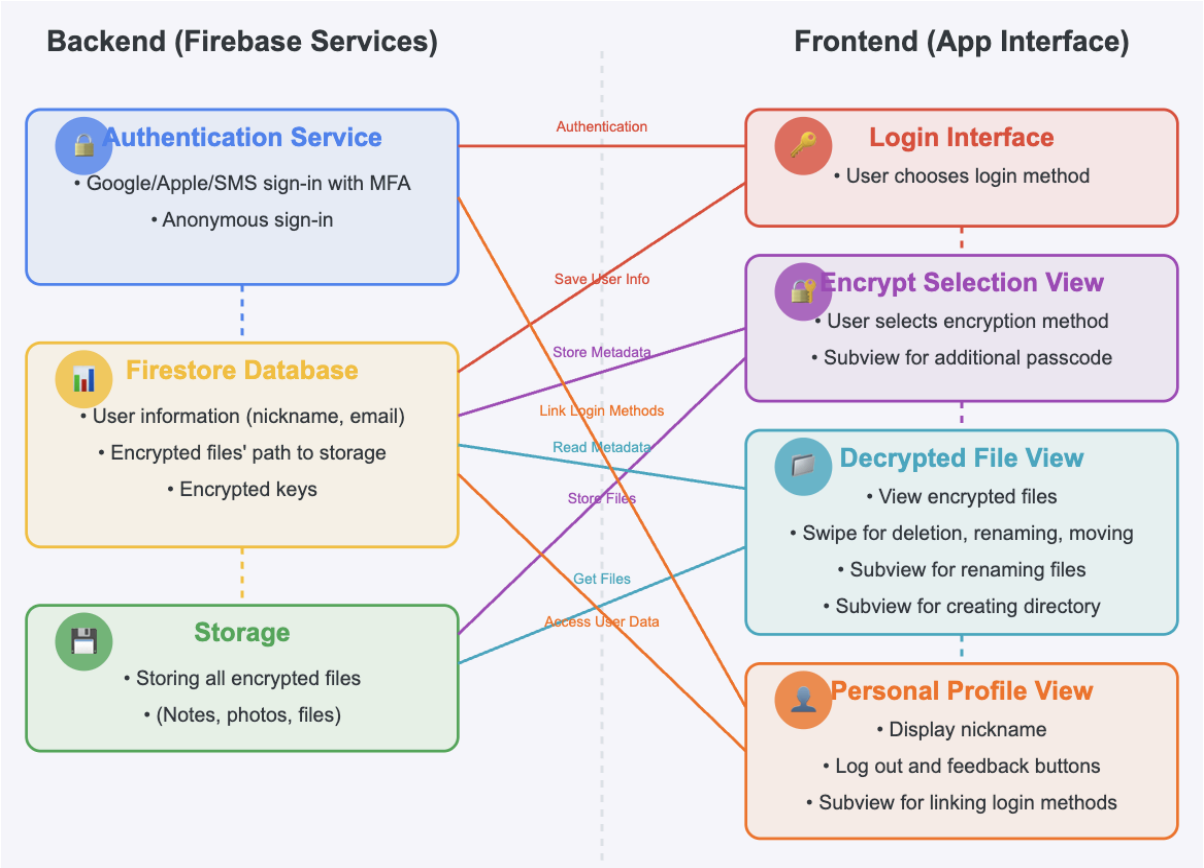
- Phone thieves cannot be protected against.
- Shoulder surfers cannot be protected against.
- Ill-intended people whom the user trusts with the information in the documents (ex. SSN) cannot be protected against.
    - This cannot be protected because we cannot limit everyone's freedom
- Social Engineering Attacks:
    - This cannot be protected because we don't have this type of technology to prevent such high-level attacks.

Section 4. Design

Overview of the design diagram:

As the system diagram shows, our app contains two parts. The first part is Firebase services which is the backend consisting of three key components: Authentication Service (Google/Apple/SMS sign-in with MFA and also the anonymous signin), Firestore database(store information for each user like users' nickname, email login and their encrypted files' path to storage and their encrypted key…), Storage(Storing all of the encrypted files(Notes, photo, files)). These components work together to ensure that encrypted files remain secure, intact, login process is secure and encrypted files.

The second part is the app interface which is the frontend also consisting of five key components: Login Interface(User chooses one way to login), Encrypt selection view(User selects one way to encrypt) having a subview for choosing additional passcode, Decrypted file view(See the files they encrypt, swiping for deletion, renaming and move directory) containing a subview for renaming the files and a subview for creating directory, Personal profile view(display nickname and give log out) containing a subview for linking another login method.



Section 4.1.

Section 4.1 How Goals Are Met

Section 4.1.1. Functionality goals met

1. **Secure Login Flow:**
   As the diagram shows, Firebase authentication service will provide support for user to login securely
2. **Secure storage of documents:**
   The diagram doesn't directly show how we set the encryption algorithm but it shows when we finish setting up the encryption algorithm, it will save the key to firestore and encrypted content to storage. In this way, our files contents and key are protected.
3. **Take/choose Photos, write note, choose files**

The diagram shows that users can easily choose one way in the encrypt selection view to do encryption.

4. **Organizational flexibility. Renaming the encrypted files, Complete deletion for files.** As the diagram shows, these functions are provided in decrypt file view and communicate with storage and firestore database to achieve these functionality

5. **Additional Security for files:** As the diagram shows, encrypt selection view has the subview for this and it will also communicate with storage and firestore database to implement the two layer protection

6. **Feedback flow, Additional security for login.** As the diagram shows, the personal profile view contains these functions and communicate with firestore database to submit feedback or gain support from authentication service to link another login method

The other functionality like user-friendly interface or smooth functionality cannot be shown in the diagram but are present in our app!

Section 4.1.2. Security goals met

1. Exclusive Document access
    a. As the diagram shows, the user needs to login and type the passcode to retrieve the file. The process is exclusive.
2. Firebase login security:
    a. Firebase service in the diagram will provide secure login protocols https://firebase.google.com/docs/auth here.
3. Strong and Secure Encryption for Documents:
    a. Not shown in the diagram, but it will be shown in the actual detailed encryption code
4. Firebase Data Storage Security:
    a. As the diagram shows, files are stored in firebase storage, and we can design the security rules for the storage space and the detailed instructions and how it works is here: https://firebase.google.com/docs/storage/security
5. No leaking personal information:
    a. As in the whole diagram, we only ask about user's email in login process and user can also choose not to tell us anything using anonymous login

Section 4.2 Known Security and Other Issues

Shoulder Surfers This vulnerability will not be protected in our application at this point because we cannot reduce amounts of bad people

Physical thieves We cannot prevent that because this is related to the country's security, which is not controlled by us.
The parties that the information is shared with We cannot prevent this because we cannot know if another person is good or bad
Apple/Google/phone number compromised We cannot prevent the shutdown or information leaking in apple, google or phone number providers like AT&T because they are the companies not owned by us.
Document Deletion

- **Incomplete Deletion**
  - Some of the info of files inevitably remain in firebase
- **Metadata Persistence**
  - While the encrypted document content is deleted, some metadata might persist in system logs or access records. (We may have ways to control it like developing a caching system to store these metadata and reproduce the info if necessary)

Section 4.3. Benefits, Harms, and Ethics Analysis

Section 4.3.1. Benefits (Assuming No Vulnerabilities)

1. **Secure document storage.** Users will benefit from having a safe and encrypted way to store their travel documents in a quick and easily accessible place: their phone.
2. Additional secure protection: we provide two-layer protection which will give users a great sense of security for sure.
3. **Efficient verification process.** Transportation employees will have an easier time verifying information because the client will have quicker access using the app.
4. **Convenient documents organization.** Our app allows users to efficiently manage and save their own important docs efficiently
5. **Prevent forgetting documents.** Our app will provide an alternate storage space for documents that will be useful in some situations

Section 4.3.2. Harms (Assuming No Vulnerabilities)

1. Competitor apps that provide similar services may lose users who change platforms.
2. There is potential that the app may slow down users, as they need to enter a password and PIN to unlock the app and access files
3. Users may lose access to their account or documents if they forget their login
4. Users may now highly depend on the digital device, so they will not be able to keep the physical copy of the important documents
5. Users may need to choose between more security with more to remember and less security with less to remember.

Section 4.3.3. Benefits (If One or More Vulnerabilities Exist)

1. Still multiple security layers. Although there will be physical security like shoulder surfing or authentication compromises. There are still multiple steps for viewing the saved documents, which increases the security.
2. Assuming the encryption function does not work as well as intended or the pin is insecure, the function to scan, upload, and store documents is still beneficial.
3. Assuming the login function does not work well, the encryption will still keep the documents safe. Because our app is like two-layer protection or even more layer if you set additional passcodes. So even if anyone can log into your account, people still cannot decrypt files if they don't know the passcode.
4. Offline access security. Assuming the vulnerability exists like I mentioned before, but if the travelers are in some remote countries., which cannot connect to our Firebase through the Internet, then his docs are still stored securely. But in this case, it is also a harm because you cannot access the file either.

Section 4.2.4. Harms (If One or More Vulnerabilities Exist)

1. Shoulder surfing: people may directly see the sensitive information in your doc and they may spread the info with others
2. Physical Device: not only your files but also your login information and apple information will be leaked.
3. If a third party leaked your information: they can log into your account, even if they cannot see the encrypted files, they may change your passcode and make you unable to access your account in the future.
4. If for example google was shut down, then users with google login will lose their files immediately.
5. Incomplete deletion. As the complete deletion cannot be achieved, some sensitive information we choose to delete may still be kept somewhere in Firebase or Internet, which is also a threat to personal information.

Section 4.3.5 Framework Analysis

For this project, the Menlo report is best suited as it covers the issues of security, autonomy, and accessibility that are important for our app's function.

Question 1:
As our project is focused on the security of individuals, we must make sure that users understand what is happening to the data that they entrust the app with. We have to tow the line between complete accuracy in what we are doing(naming the type of encryption, what is happening during encryption, etc) as well as ensuring that users will read it and not just skip

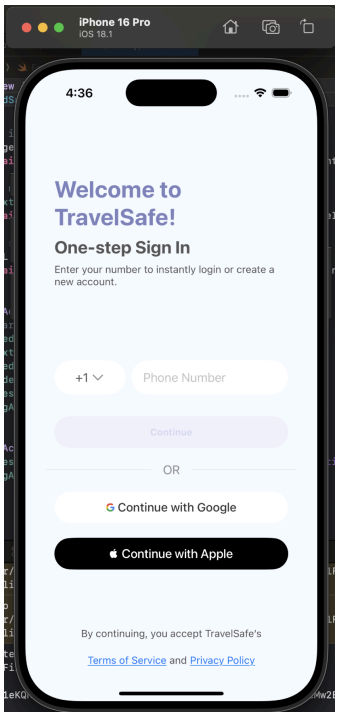
over it because it is complicated. This becomes even more of an issue when working with other systems like Apple and Google login, as we can't control how they outline their explanations of data handling. This leads us to the questions: How can we be transparent in our methods while making sure the users can understand the terminology? How can we ensure this when working with 3rd party systems(apple/google)?
Answer 1:
To ensure that users are both getting a comprehensive understanding of what is happening to their documents we can have tiers of explanations. We can offer a comprehensive explanation in our Privacy statement page. For other questions our FAQ page explains each step of the process. As for the 3rd party systems, we mention to our users that they can look at their statements because it is out of our scope.

Question 2:
A valuable piece of information for users could be the titles of their folders and files. However, they also need to know what they are trying to unlock before they do so for easy access. How can we allow users to both have autonomy over naming their files whatever they want while also informing them that the name itself could be a piece of data that should be protected?
Answer 2:
Users should be allowed to name their files whatever they want. They should be the ones determining what they think should be secure and what can be less secure. However, we can set the default name to just the date so it is not clear what is in the file.
Question 3:
We need to ensure that our app is as accessible as possible as it is an important tool for all types of people. One of our use cases even included people whose family/caregivers could load documents in the app for them for easy and safe access later. Because of the wide range of needs for different users, how can we make sure this app is accessible to the people who use it?
Answer 3:
We can make log in very easy if the user chooses to do Apple/email/phone login because no additional passwords are needed. We help users on first login understand our system by adding a tutorial. Passwords are short PINs, which hopefully will make them easy to remember.

How to address our concerns:
In our analysis of our current app we have addressed the ways we fixed concerns raised by the Menlo Framework. One of the major themes that came up was making the app more accessible and perhaps guiding the user more than we currently do. We want the user to have autonomy and not to listen to just our idea of what best practices are, but of course it is good to inform them on their first use what we think a secure way of using our app looks like. While adjusting our UI, we will continue to look at resources for what an accessible app looks like (colors, fonts, other tools) so that this is open and usable to as many people as possible. We will also look at other ways websites and apps describe and present data use and privacy to their users so it is both comprehensive and easy to understand.

Section 5. Implementation

Section 5.1. Running Your Project



Early Stage(not posted to apple store):

If You Use Mac:

1. Be invited to our github repo, you can contact Shawn through email: ms6592@columbia.edu
2. Download the xcode(not xcode beta) through the following link: https://developer.apple.com/xcode/
3. Go to our github repo: https://github.com/Aristotle2003/TravelSafe and copy this link
4. Open Xcode, by clicking on Clone Git Repository and then paste the link in the second step and it will prompt you to log into your github account and it will successfully clone the full repo in xcode
5. Xcode will direct you to the editing view and in the top row, you can choose your running destination, you may not have one at this moment, but you can download the stimulator at this point or you can download the stimulator earlier along with downloading xcode
6. Finally, you can now choose the phone in your stimulator to run with, and you can click on the start button in the left top corner of the view and wait for a few moments. Your mac will display an iphone as follows and you can click to experience our app.

If you don't have Mac but have iphone/ipad:

1. You can contact us and we can manually install the app on your phone using a iphone cable connecting with Mac, and then you can run and test our app
2. The second way is that we may release test version so that you can directly download through testflight

If you are not fan of Apple:

1. You can contact us and we can give you other alternative options. For example: we can lend you our devices to explore our app, or we can show you demos of our app.

Late Stage: (We are here now!!! But it may Apple a week to review our app, so the link may not work now)

1. Our app is reviewed by apple and posted to testfligh for testing!!
2. So if you don't want to download the app but want to experience our app. You can just open the link https://testflight.apple.com/join/Nk8xvWXA to directly download on your

phone. (If you didn't use testflight before, you may need to download testflight first and reclick on the link!!!)
3. When we finish collecting enough feedback, we will distribute it to app store(just a one-click from now)

Section 5.2. Implementation Security

Section 5.2.1. Achieved Security Properties

1. The function firstly takes the passcode user entered and converts the passcode to the form UTF-8 using the function .data(using: .utf8)
2. Then we generate a salt called "EncryptionSalt123" and also use the function .data(using: .utf8) to convert it to the form UTF-8 as well. And we generate a symmetric key for our passcode and then combine the key with the salt and hash using SHA256. The resulting hash value becomes our actual encryption key in our system.
3. Then we generate a nounce using AES.GCM.Nounce and this ensures even if the same data is encrypted multiple times with the same key, the output will still be different
4. Then we use AES.GCM to encrypt the key and accordingly nounce, and we set the result to be sealedBox containing our encrypted data.
5. Finally, we have a combined data to record the nounce, the encrypted data and the authentication tag for the decryption process.

```
Unset
import CryptoKit

func encryptContent(_ data: Data, withPasscode passcode: Int) throws ->
Data {
        let passcodeString = String(format: "%03d", passcode)
        guard let passcodeData = passcodeString.data(using: .utf8) else
{
            throw EncryptionError.invalidPasscode
        }
        let salt = "EncryptionSalt123".data(using: .utf8)!
        var key = SymmetricKey(data: passcodeData)
        let keyData = SHA256.hash(data: key.withUnsafeBytes { Data($0) }
+ salt)
        key = SymmetricKey(data: keyData)
        let nonce = try AES.GCM.Nonce()
        let sealedBox = try AES.GCM.seal(data, using: key, nonce: nonce)
        var combinedData = Data()
```

```
        combinedData.append(nonce.withUnsafeBytes { Data($0) })
        combinedData.append(sealedBox.ciphertext)
        combinedData.append(sealedBox.tag)
        return combinedData
    }
```

Why are we confident our algorithm is secure?

1. The whole encryption process is successful by testing
2. We cannot download and see the encrypted datas in Firebase, proving that it is a secure storage.
3. We also test the decryption process that only the correct passcode can see the file meaning our encryption key is stored correctly and the encryption process is also correct

Preprocessing for files and images, for large images and files, we also provide the method to transform it to form like notes for encryption, the code is as follows: so for images we use jpegData to make it become image data we can encrypt, and for files, we use .mappedIfSafe to map content of files for encryption

Unset

```
// MARK: - Image Encryption
    private func encryptImage(_ image: UIImage, withPasscode passcode: Int) throws
-> Data {
        guard let imageData = image.jpegData(compressionQuality: 0.7) else {
            throw EncryptionError.encryptionFailed
        }
        return try encryptContent(imageData, withPasscode: passcode)
    }

    // MARK: - File Encryption
    private func encryptFile(_ fileURL: URL, withPasscode passcode: Int) throws ->
Data {
        do {
            let fileData = try Data(contentsOf: fileURL, options: .mappedIfSafe)
            return try encryptContent(fileData, withPasscode: passcode)
        } catch {
            throw EncryptionError.encryptionFailed
        }
    }
```

The login implementation: (I will use a sample of google login, other methods will use similar way to communicate with firebase and authenticate users) So the google sign in begins with retrieving Firebase ID and create a connection between google and our firebase. When a user initiates login, the app will show google login UI and enters their account. When the Firebase receives tokens upon their successful login, the firebase will use the token to create firebase credentials for the user. And later firebase will use this credentials to authenticate the user.

Unset

```
// Sign in using Google sign in
    private func handleGoogleSignIn() {
        guard let clientID = FirebaseManager.shared.auth.app?.options.clientID else
{
            self.loginStatusMessage = "Error getting client ID"
            return
        }

        // Create Google Sign In configuration object.
        let config = GIDConfiguration(clientID: clientID)
        GIDSignIn.sharedInstance.configuration = config

        guard let windowScene = UIApplication.shared.connectedScenes.first as?
UIWindowScene,
              let rootViewController = windowScene.windows.first?.rootViewController
else {
            self.loginStatusMessage = "Error getting root view controller"
            return
        }

        // Start the sign in flow!
        GIDSignIn.sharedInstance.signIn(withPresenting: rootViewController) {
result, error in
            // User cancel the google UI
            if let error = error {
                self.loginStatusMessage = "Error signing in with Google:
\(error.localizedDescription)"
                return
            }


            guard let user = result?.user,
                  let idToken = user.idToken?.tokenString
            else {
                self.loginStatusMessage = "Error getting user data"
                return
            }
```

```
            let credential = GoogleAuthProvider.credential(
                withIDToken: idToken,
                accessToken: user.accessToken.tokenString
            )



            // Sign in google user!!
            FirebaseManager.shared.auth.signIn(with: credential) { result, error in
                if let error = error {
                    self.loginStatusMessage = "Firebase sign in error:
\(error.localizedDescription)"
                    return
                }

                // Check if user exists
                guard let user = result?.user else { return }
                FirebaseManager.shared.firestore.collection("users")
                    .document(user.uid).getDocument { snapshot, error in
                        if let error = error {
                            self.loginStatusMessage = "\(error)"
                            return
                        }
                        // User exists
                        if (snapshot?.data()) != nil {
                            checkTutorialStatus()
                            self.isLogin = true

                        // New user set up profile
                        } else {
                            self.showProfileSetup = true
                        }
                    }
            }
        }
    }
```

Why are we confident?

1. Successful login for every method
2. Able to see the users registered in our firebase backend and correctly show the login method but cannot see any user personal info, which proves our login is correct and secure

Face ID security implementation:

So we implement the security check using Face ID, the code is as follows. This biometricauthservice class first determines if the device is capable of face ID. And when face id is detected, the system will return the .faceID enum case. And the actual authentication process for face ID will be achieved using autheticateWithBiometrics(reason:) function below and inside it, we have a method called evaluatePolicy to create a clean interface to handle face ID.

Unset

```
class BiometricAuthService {
    enum BiometricType {
        case none
        case touchID
        case faceID
    }

    enum BiometricError: LocalizedError {
        case authenticationFailed
        case userCancel
        case userFallback
        case biometryNotAvailable
        case biometryNotEnrolled
        case biometryLockout
        case unknown

        var errorDescription: String? {
            switch self {
            case .authenticationFailed: return "Authentication failed"
            case .userCancel: return "User canceled"
            case .userFallback: return "User selected fallback option"
            case .biometryNotAvailable: return "Biometry not available"
            case .biometryNotEnrolled: return "Biometry not enrolled"
            case .biometryLockout: return "Biometry locked out"
            case .unknown: return "Unknown error"
            }
        }
    }

    // Check what biometric authentication is available
    func biometricType() -> BiometricType {
        let context = LAContext()
        var error: NSError?

        guard context.canEvaluatePolicy(.deviceOwnerAuthenticationWithBiometrics,
error: &error) else {
            return .none
        }

        if #available(iOS 11.0, *) {
```

```
            switch context.biometryType {
            case .none:
                return .none
            case .touchID:
                return .touchID
            case .faceID:
                return .faceID
            @unknown default:
                return .none
            }
        } else {
            return
context.canEvaluatePolicy(.deviceOwnerAuthenticationWithBiometrics, error: nil) ?
.touchID : .none
        }
    }

    // Authenticate using biometrics
    func authenticateWithBiometrics(reason: String) async throws -> Bool {
        return try await withCheckedThrowingContinuation { continuation in
            let context = LAContext()

            context.evaluatePolicy(.deviceOwnerAuthenticationWithBiometrics,
localizedReason: reason) { success, error in
                if let error = error {
                    let biometricError: BiometricError
                    switch error {
                    case LAError.authenticationFailed:
                        biometricError = .authenticationFailed
                    case LAError.userCancel:
                        biometricError = .userCancel
                    case LAError.userFallback:
                        biometricError = .userFallback
                    case LAError.biometryNotAvailable:
                        biometricError = .biometryNotAvailable
                    case LAError.biometryNotEnrolled:
                        biometricError = .biometryNotEnrolled
                    case LAError.biometryLockout:
                        biometricError = .biometryLockout
                    default:
                        biometricError = .unknown
                    }
                    continuation.resume(throwing: biometricError)
                } else {
                    continuation.resume(returning: success)
                }
            }
```

```
        }
    }
}
```

All of our other implementations are in github and there are a lot of interesting functionalities mentioned above(have comments in code), please take a look!!

Section 5.2.2. Known Issues

What are known issues with your implementation?

1. Weak Password Entropy: a 3-digit passcode only has 1000 possible combinations, which is highly vulnerable to brute force attacks
    a. Maybe require a longer and more complex password
2. Static Salt usage: the salt is hard-coded as ”EncryptionSalt123”, meaning it is the same for every encryption operation.
    a. Generate a random salt for each encryption process
3. SHA256 is not computationally intensive, meaning that people can still use computer to brutally try every possible hashing results to attack the passcode
    a. Strong functions like PBKDF2 or Argon2 which will prevent brutal force
4. The login process may leak personal information like email or phone number
    a. We have already solved that by giving an anonymous login option

We are enhancing the security more by providing a second layer of security mentioned before.

Section 6. Usability Evaluation.

Section 6.1 Peer evaluation

Our peer group is the file transfer group and the below is the summary of their suggestions. The laws of UX we are aligned with are Hick’s law which is simple to use and Law of Uniform Connectedness which is categorizing similar groups together and Tesler’s law which is a clean app without any ambiguous icons. The laws of UX that you didn’t align with are flow which doesn’t provide feedback(implemented the button now), paradox of the active users(implemented the tutorial now) and Miller’s law because there are a little too many things

to remember. They have several recommendations: some UI fixes, the bugs for navigation, the renaming functionality, unclear about how to create the self-customized directory.

Section 6.2 Cognitive walkthrough

Section 6.2.1. Set-up

Goals:
Our goal was for the user to successfully be able upload, encrypt, and save files onto their phone. Upon opening the app, we asked the user to make an account with the method of their choice (phone number, google, apple). Afterwards they were asked to do the following features:

- upload files using the 4 options (“take photo”, “upload photo”, “upload file”, “write note”),
- encrypt and save the files
- view the files in their directory/folders
- logout and log back into the same account

We chose these goals since they are the most basic features of the app, the user’s ability to complete the tasks would help us gauge how to implement the features better.

Section 6.2.2. Cognitive walkthrough results

Here, discuss the results of your cognitive walkthroughs and the mini-interview.
During the walkthrough: What was easy for them? What was hard for them? What did they get stuck on? Is there anything they thought they understood but didn’t? Did they find any bugs?
Afterwards: What did they think of the system? Would they use it? Did they understand the connection to security?

We conducted 4 interviews. Overall, all users found the application straightforward and easy to navigate. The four functionalities (login, upload, view, and logout) were mostly completed without any challenges. For login, 3 out of 4 users opted for phone number login which seemed easiest because it was a single verification code needed to set up the account. When in the app, the different methods to upload files were clearly indicated on the screen, users understood what each button meant. The participants enjoyed the “look” of our app (color, font, spacing), although there were some small visual things that needed to be fixed (changing button color to indicate that you can press it now, hiding password with asterisks, etc). One of the biggest takeaways from this walkthrough was that although people had heard of encryption, and therefore understood the link between our app and security, they did not really understand what it meant or how it made their documents secure. After explaining what encryption and how it makes files secure, users mentioned they were more inclined to use it to store important information as opposed to the default Notes app on their phone.

Section 6.2.3. System Usability Scale results

From the system's usability google form, we found that all users responded relatively similar to each other. Below are the results for each question on the survey:

- “I think that I would like to use this system frequently”: between 3-4, more likely to use after an explanation of what encryption was and how it protected their security
- “I found the system unnecessarily complex”: 1, users noted that the application was very straightforward
- “I thought the system was easy to use”: 5
- “I think that I would need the support of a technical person to be able to use the system”: 1, this again relates to the previous two questions. Users were able to easy navigate through the app without assistance
- “I found the various functions in this system were well integrated”: 4-5, issues with getting set up and saving names of the notes
- “I thought there was too much inconsistency in this system”: 1
- “I would imagine that most people would learn to use this system very quickly”: 4-5
- “I found the system very cumbersome to use”: 1
- “I felt very confidence using the system”: 5
- “I needed to learn a lot of things before I could get going with this system”: 1-2, users noted that knowing the purpose of this app is helpful

Section 6.2.4 Take-aways

1. People say that they may not use this system frequently because of the lack of tutorials. So, we will add more tutorial pages for clearer guidance. And maybe we can keep the button for displaying tutorials after the first time to prevent users from forgetting
2. Based on surveys and interviews, people all think the app is straightforward and easy to use which is good.
3. Some functionalities are not very clear like directories and renaming, so we must provide a tutorial or another visible UI
4. Better looking UI to attract more as interviewee may think the app is a bit too normal

References

[1] Apple Inc. (2024.). *Notes* (iOS 18). App Store. https://apps.apple.com/us/app/notes/id1110145109

[2] Brockway, G et al (2025). *TripIt: Travel planner*. (v19.5) AppStore. https://www.tripit.com

[3] Readdle Technologies Limited (2025). *Documents: File Manager & Docs.* (v18.16.6)

AppStore. https://apps.apple.com/us/app/documents-file-manager-docs/id364901807

For code: original, no reference or use api from any source code. But we import some packages downloaded in xcode. The citation for these packages are listed here:

[4] Apple Inc. (2019). SwiftUI Framework. Apple Developer Documentation.
https://developer.apple.com/documentation/swiftui

[5] Apple Inc. (2019). CryptoKit Framework. Apple Developer Documentation.
https://developer.apple.com/documentation/cryptokit

[6] Apple Inc. (2019). Authentication Services Framework. Apple Developer Documentation.
https://developer.apple.com/documentation/authenticationservices

[7] Apple Inc. (2014). Local Authentication Framework. Apple Developer Documentation.
https://developer.apple.com/documentation/localauthentication
[8] Apple Inc. (2017). PDFKit Framework. Apple Developer Documentation.
https://developer.apple.com/documentation/pdfkit

[9] Apple Inc. (2008). QuickLook Framework. Apple Developer Documentation.
https://developer.apple.com/documentation/quicklook

[10] Google LLC. (2023). Firebase iOS SDK. Firebase Documentation.
https://firebase.google.com/docs/ios/setup

[11]] Google LLC. (2023). Firebase Authentication. Firebase Documentation.
https://firebase.google.com/docs/auth

[12] Google LLC. (2023). Firebase Cloud Storage. Firebase Documentation.
https://firebase.google.com/docs/storage

[13] Google LLC. (2023). Cloud Firestore. Firebase Documentation.
https://firebase.google.com/docs/firestore

[14] Google LLC. (2023). Google Sign-In for iOS. Google Developers.
https://developers.google.com/identity/sign-in/ios

Acknowledgements

The acknowledgement for this template document (which you do not need to include in your report): This template document has been adapted from the design document used by the University of Washington's Computer Security Capstone course (UWCSE481S), which was

created by Tadayoshi Kohno and Franziska Roesner. Lucy Simko has adapted it to fit Barnard's Building Usable Security course, with permission. The original document, and more information about the UW course, can be found here: https://security-education.cs.washington.edu/

Here are the original acknowledgements from this document:
This template document was originally created by Tadayoshi Kohno and Franziska Roesner. We thank past TAs Kiron Lebeck and Lucy Simko for suggestions and edits. We also thank Megan Finn and Justin Petelka for contributions to the ethics section.

We are so grateful to Professor Simko for helping us along the way and also very grateful to all of the people in class who provide a lot of valuable suggestions after every presentation, especially the File Transfer group for their dedicated peer analysis. Finally, we would like to thank our four interviewees for sitting with us and giving us valuable feedback to make our app better!